```java
public class BinaryTree<ELEMENT> {


    //region Binary Tree Node Class

    protected class BTNode<ELEMENT> {

        public ELEMENT item;
        public BTNode<ELEMENT> left;
        public BTNode<ELEMENT> right;

        public BTNode() {
            this(null, null, null);
        }
        public BTNode(ELEMENT item) {
            this(item, null, null);
        }
        public BTNode(ELEMENT item, BTNode<ELEMENT> left, BTNode<ELEMENT> right) {
            this.item = item;
            this.left = left;
            this.right = right;
        }

        @Override
        public String toString() {
            return this.item.toString();
        }

        // Método para propósitos académicos
        public void Visit() {
            System.out.printf("%s ", this.item.toString());
        }
    }
    //endregion





    //region Attributes

    protected BTNode<ELEMENT> root;

    //endregion



    //region Constructors
```

```java
public BinaryTree() {
   this.root = null;
}

// Métodos para propósitos académicos
public BinaryTree(ELEMENT item) {
   this(item, null, null);
}
public BinaryTree(ELEMENT item, BinaryTree<ELEMENT> left, BinaryTree<ELEMENT> right) {
   this.root = new BTNode<ELEMENT>(item, null, null);
   if (left != null) {
      this.root.left = left.root;
   }
   if (right != null) {
      this.root.right = right.root;
   }
}

//endregion

@Override
public String toString() {
   StringBuilder sb = new StringBuilder();
   toString(sb, this.root);
   return sb.toString();
}
protected void toString(StringBuilder sb, BTNode<ELEMENT> root) {
   if (root != null) {
      sb.append(root.item.toString());
      if (root.left != null) {
         sb.append("(");
         toString(sb, root.left);
         if (root.right != null) {
            sb.append(",");
            toString(sb, root.right);
         }
         sb.append(")");
      } else {
         if (root.right != null) {
            sb.append("(,");
            toString(sb, root.right);
            sb.append(")");
         }
      }
   }
}



public void PreOrder() {
    PreOrder(this.root);
}
protected void PreOrder(BTNode<ELEMENT> root) {
    if (root != null) {
        root.Visit();
        PreOrder(root.left);
        PreOrder(root.right);
    }
}

public void InOrder() {
    InOrder(this.root);
}
protected void InOrder(BTNode<ELEMENT> root) {
    if (root != null) {
        InOrder(root.left);
        root.Visit();
        InOrder(root.right);
    }
}

public void PostOrder() {
    PostOrder(this.root);
}
protected void PostOrder(BTNode<ELEMENT> root) {
    if (root != null) {
        PostOrder(root.left);
        PostOrder(root.right);
        root.Visit();
    }
}

public void DescendingOrder() {
    DescendingOrder(this.root);
}
protected void DescendingOrder(BTNode<ELEMENT> root) {
    if (root != null) {
        DescendingOrder(root.right);
        root.Visit();
        DescendingOrder(root.left);
    }
}



public int NodeCount() {
    return NodeCount(this.root);
}
protected int NodeCount(BTNode<ELEMENT> root) {
    if (root != null) {
```

```java
        return 1 + NodeCount(root.left) + NodeCount(root.right);
    }
    return 0;
}



public int LeafCount() {
    return LeafCount(this.root);
}
protected int LeafCount(BTNode<ELEMENT> root) {
    if (root != null) {
        if ( (root.left == null) && (root.right == null) ) {
            return 1;
        } else {
            return LeafCount(root.left) + LeafCount(root.right);
        }
    }
    return 0;
}



public int InternalCount() {
    return InternalCount(this.root);
}
protected int InternalCount(BTNode<ELEMENT> root) {
    if (root != null) {
        if ( (root.left == null) && (root.right == null) ) {
            return 0;
        } else {
            return 1 + InternalCount(root.left) + InternalCount(root.right);
        }
    }
    return 0;
}



public int MaxLevel() {
    return MaxLevel(this.root);
}
protected int MaxLevel(BTNode<ELEMENT> root) {
    if (root != null) {
        if ( (root.left != null) || (root.right != null) ) {
            int leftLevel = MaxLevel(root.left);
            int rightLevel = MaxLevel(root.right);
            return 1 + Math.max(leftLevel, rightLevel);
        }
        return 0;
    }
    return -1;
```

```
  }



  public int Height() {
    return MaxLevel() + 1;
  }
}
```